下标（Subscripts）

1.0 翻译：siemenliu (https://github.com/siemenliu) 校对：zq54zquan (https://github.com/zq54zquan)

2.0 翻译+校对：shanks (http://codebuild.me)

2.1 翻译+校对：shanks (http://codebuild.me)，Realank (https://github.com/Realank)

2.2 校对：SketchK (https://github.com/SketchK) 2016-05-13

本页包含内容：

下标（subscripts）可以定义在类（class）、结构体（structure）和枚举（enumeration）中，是访问集合（collection），列表（list）或序列（sequence）中元素的快捷方式。可以使用下标的索引，设置和获取值，而不需要再调用对应的存取方法。举例来说，用下标访问一个 `Array` 实例中的元素可以写作 `someArray[index]`，访问 `Dictionary` 实例中的元素可以写作 `someDictionary[key]`。

一个类型可以定义多个下标，通过不同索引类型进行重载。下标不限于一维，你可以定义具有多个入参的下标满足自定义类型的需求。

下标语法

下标允许你通过在实例名称后面的方括号中传入一个或者多个索引值来对实例进行存取。语法类似于实例方法语法和计算型属性语法的混合。与定义实例方法类似，定义下标使用 `subscript` 关键字，指定一个或多个输入参数和返回类型；与实例方法不同的是，下标可以设定为读写或只读。这种行为由 getter 和 setter 实现，有点类似计算型属性：

```
subscript(index: Int) -> Int {
    get {
      // 返回一个适当的 Int 类型的值
    }

    set(newValue) {
      // 执行适当的赋值操作
    }
}
```

`newValue` 的类型和下标的返回类型相同。如同计算型属性，可以不指定 setter 的参数（`newValue`）。如果不指定参数，setter 会提供一个名为 `newValue` 的默认参数。

如同只读计算型属性，可以省略只读下标的 `get` 关键字：

```
subscript(index: Int) -> Int {
    // 返回一个适当的 Int 类型的值
}
```

下面代码演示了只读下标的实现，这里定义了一个 `TimesTable` 结构体，用来表示传入整数的乘法表：

```
struct TimesTable {
    let multiplier: Int
    subscript(index: Int) -> Int {
        return multiplier * index
    }
}
let threeTimesTable = TimesTable(multiplier: 3)
print("six times three is \(threeTimesTable[6])")
// 输出 "six times three is 18"
```

在上例中，创建了一个 `TimesTable` 实例，用来表示整数 3 的乘法表。数值 3 被传递给结构体的构造函数，作为实例成员 `multiplier` 的值。

你可以通过下标访问 `threeTimesTable` 实例，例如上面演示的 `threeTimesTable[6]`。这条语句查询了 3 的乘法表中的第六个元素，返回 3 的 6 倍即 18。

> 注意
>
> `TimesTable` 例子基于一个固定的数学公式，对 `threeTimesTable[someIndex]` 进行赋值操作并不合适，因此下标定义为只读的。

下标用法

下标的确切含义取决于使用场景。下标通常作为访问集合（collection），列表（list）或序列（sequence）中元素的快捷方式。你可以针对自己特定的类或结构体的功能来自由地以最恰当的方式实现下标。

例如，Swift 的 `Dictionary` 类型实现下标用于对其实例中储存的值进行存取操作。为字典设值时，在下标中使用和字典的键类型相同的键，并把一个和字典的值类型相同的值赋给这个下标：

```
var numberOfLegs = ["spider": 8, "ant": 6, "cat": 4]
numberOfLegs["bird"] = 2
```

上例定义一个名为 `numberOfLegs` 的变量，并用一个包含三对键值的字典字面量初始化它。 `numberOfLegs` 字典的类型被推断为 `[String: Int]` 。字典创建完成后，该例子通过下标将 `String` 类型的键 `bird` 和 `Int` 类型的值 `2` 添加到字典中。

更多关于 `Dictionary` 下标的信息请参考读取和修改字典 (./04_Collection_Types.html#accessing_and_modifying_a_dictionary)

> 注意
>
> Swift 的 `Dictionary` 类型的下标接受并返回可选类型的值。上例中的 `numberOfLegs` 字典通过下标返回的是一个 `Int?` 或者说“可选的int”。 `Dictionary` 类型之所以如此实现下标，是因为不是每个键都有个对应的值，同时这也提供了一种通过键删除对应值的方式，只需将键对应的值赋值为 `nil` 即可。

下标选项

下标可以接受任意数量的入参，并且这些入参可以是任意类型。下标的返回值也可以是任意类型。下标可以使用变量参数和可变参数，但不能使用输入输出参数，也不能给参数设置默认值。

一个类或结构体可以根据自身需要提供多个下标实现，使用下标时将通过入参的数量和类型进行区分，自动匹配合适的下标，这就是*下标的重载*。

虽然接受单一入参的下标是最常见的，但也可以根据情况定义接受多个入参的下标。例如下例定义了一个 `Matrix` 结构体，用于表示一个 `Double` 类型的二维矩阵。 `Matrix` 结构体的下标接受两个整型参数：

```
struct Matrix {
    let rows: Int, columns: Int
    var grid: [Double]
    init(rows: Int, columns: Int) {
        self.rows = rows
        self.columns = columns
        grid = Array(count: rows * columns, repeatedValue: 0.0)
    }
    func indexIsValidForRow(row: Int, column: Int) -> Bool {
        return row >= 0 && row < rows && column >= 0 && column < columns
    }
    subscript(row: Int, column: Int) -> Double {
        get {
            assert(indexIsValidForRow(row, column: column), "Index out of range")
            return grid[(row * columns) + column]
        }
        set {
            assert(indexIsValidForRow(row, column: column), "Index out of range")
            grid[(row * columns) + column] = newValue
        }
    }
}
```

`Matrix` 提供了一个接受两个入参的构造方法，入参分别是 `rows` 和 `columns` ，创建了一个足够容纳 `rows * columns` 个 `Double` 类型的值的数组。通过传入数组长度和初始值 `0.0` 到数组的构造器，将矩阵中每个位置的值初始化为 `0.0` 。关于数组的这种构造方法请参考创建一个空数组 (./04_Collection_Types.html#creating_an_empty_array)。

你可以通过传入合适的 `row` 和 `column` 的数量来构造一个新的 `Matrix` 实例：

```
var matrix = Matrix(rows: 2, columns: 2)
```

上例中创建了一个 `Matrix` 实例来表示两行两列的矩阵。该 `Matrix` 实例的 `grid` 数组按照从左上到右下的阅读顺序将矩阵扁平化存储：

将 row 和 column 的值传入下标来为矩阵设值，下标的入参使用逗号分隔：

```
matrix[0, 1] = 1.5
matrix[1, 0] = 3.2
```

上面两条语句分别调用下标的 setter 将矩阵右上角位置（即 row 为 0 、 column 为 1 的位置）的值设置为 1.5 ，将矩阵左下角位置（即 row 为 1 、 column 为 0 的位置）的值设置为 3.2 ：

Matrix 下标的 getter 和 setter 中都含有断言，用来检查下标入参 row 和 column 的值是否有效。为了方便进行断言， Matrix 包含了一个名为 indexIsValidForRow(_:column:) 的便利方法，用来检查入参 row 和 column 的值是否在矩阵范围内：

```
func indexIsValidForRow(row: Int, column: Int) -> Bool {
    return row >= 0 && row < rows && column >= 0 && column < columns
}
```

断言在下标越界时触发：

```
let someValue = matrix[2, 2]
// 断言将会触发，因为 [2, 2] 已经超过了 matrix 的范围
```

上一篇：方法 (/project/swift/chapter2/11_Methods.html)

下一篇：继承 (/project/swift/chapter2/13_Inheritance.html)

Via 由 [极客学院 Wiki (http://wiki.jikexueyuan.com)] 提供

Realank刘 (http://weibo.com/realank)
回复 dong: 说得有道理，明天我来改
(http://weibo.com/realank)1月18日 回复 顶 转发

夜游
回复 东东: no can no bb, you can you up
1月4日 回复 顶 转发

dong (http://weibo.com/wenbokenet)
感谢译者的辛勤努力和付出，这里翻译成「下标」确实更好些，希望能采纳，辛苦了
(http://weibo.com/wenbokenet)1月4日 回复 顶(2) 转发

rayco (http://t.qq.com/zr_1209)
Subscripts就翻译成"下标"不是挺好的吗
(http://t.qq.com/zr_1209)2015年12月31日 回复 顶 转发

阿喀琉斯
首先要谢谢译者
2015年12月30日 回复 顶 转发

cao
回复 东东: 不看可以gun
2015年12月22日 回复 顶 转发

G梅果果 (http://weibo.com/2063322140)
看见评论都好心寒,中国开发界的文档一直不完备,终于有了这些无偿提供翻译的作者,结果读者还嫌弃人家翻译不准,嫌不准自己去看原版文档啊没人求你们来这里,我不会嘲笑你们英语烂的掉渣的,就这样的素质,不会是个合格的开发者,合格的员工的,因为人从根上就长歪了
(http://weibo.com/2063322140)2015年12月22日 回复 顶(9) 转发

youke
回复 东东: 有建议可以提,冷嘲热讽作者?自己去翻译啊
2015年12月17日 回复 顶 转发

东东 (http://t.qq.com/f0u1c2k3G4F5W)
"下标脚本"是你发明的吗, 你咋不翻译成子脚本啊. 真是长见识了..
(http://t.qq.com/f0u1c2k3G4F5W)2015年12月12日 回复 顶(1) 转发

chenyf
回复 chenyf: input parameters 翻译为入参也别扭
2015年12月2日 回复 顶 转发

chenyf
Subscripts 翻译成下标脚本有点接受不了。
2015年12月1日 回复 顶 转发

舒服
加倍！谢谢作者！
2015年11月19日 回复 顶 转发

qxh
抢地主，谢谢作者~~
2015年11月16日 回复 顶 转发

鸡巴掉了
强jb,谢谢作者
2015年11月13日 回复 顶 转发